JAVA Naming Conventions

1. Introduction As JAVA is increasingly used within the DII COE, the potential for JAVA classes to have conflicting names increases. This paper explains the potential problem and provides a simple naming convention to prevent these conflicts.

1.1 Classes and Objects An object is defined by it class, which describes everything that objects of that class know (its attributes) and can do (its methods).

Important Definitions
Objects are individual instances of a class. A class is a template or blueprint from which an object is actually constructed. For example, we might have an Employee class, with attributes like dob, salary, job title, and methods like raiseSalary(), hireYear(), getName() and print().

In Java, we can create a new Employee object or instance this way:

```
Employee staff = new Employee();
```

In the above example a new object was created by first using a preexisting class (Employee), *declaring* it with a new name (staff) and then creating a new *instance* of it[1].

1.2 Java File names and Directory Structure A Java source code file has the extension *.java*. Each class definition in a .java file is compiled into a separate "class file" with the extension *.class*[2].

1.3 Package A package is a group of related Java classes or interfaces. When you are writing Java code, you can put a class or multiple classes into a package. After classes are written and packaged, you can call on classes in a package by using an import statement.

All files of a package must be located in a subdirectory that matches the full package name. For example, all files in the Java.util package are in a subdirectory java/uit/ (Unix) or java/util (windows)[3].

1.4 CLASSPATH The CLASSPATH command tells the compiler where to start looking for a package in the file system.

1.5 Import The import command allows programmers to use the names of classes directly by "importing" just the name of the full package[4].

As a result, the import statement makes Java classes accesable using an abreviated name (of

course Java classes can always be referred to by their fully qualified name).

Without using the import command, we might write the following Java code:

```
that.really.long.package.name.ClassOne      first;
that.really.long.package.name.ClassTwo      second;
that.really.long.package.name.ClassThree    third;
```

This code declares references to ClassOne, ClassTwo, and ClassThree which all belong to the package *that.really.long.package.name*.

Now using the import command, we can write:

```
import that.really.long.package.name.*;

ClassOne      first;
ClassTwo      second;
ClassThree    third;
```

2.<u>The Problem</u> When two classes with the same name are imported into the same applet or application a conflict occurs.

The graphic below shows a notional directory/file system where two class libraries are stored, one in *h:\seg1\classes* and one in *h:\seg2\classes*. We want to follow the Java community convention of putting class files into a directory named *classes*.

When a developer uses the above directory/file system the following problem can occur:

In UNIX the classpath would be set as:
setenv CLASSPATH = c:/h/seg1/classes; c:/h/seg2/classes

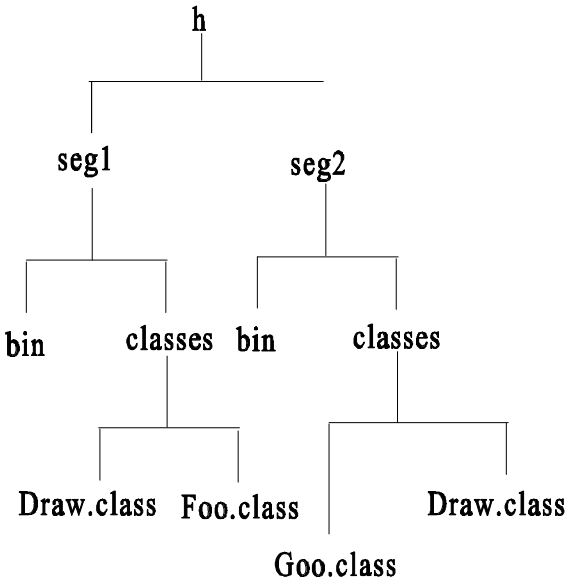
In Windows NT/95:
set CLASSPATH = c:\h\seg1\classes; c:\h\seg2\classes

Now, consider the following Java code:

```
//declaring and instantiating
Draw draw_obj1 = new Draw();      //**PROBLEM - which Draw class??**
Draw draw_obj2 = new Draw();      //**PROBLEM - which Draw class??**
Foo myfoo = new Foo();            // OK, no conflict
Goo yourgoo = new Goo();          // OK, no conflict
```

When we start to declare and instantiate objects, we run into trouble, since we cannot differentiate between the Draw class in seg1\classes and the Draw class in seg2\classes.

3. <u>Proposed Solution</u> By using the full class name of the segment (not the segment prefix) within



the directory structure we can provide a unique name for each Draw class.

The code from the first example can now be modified to use the unique names for each of the Draw classes. The CLASSPATHs are set as before:

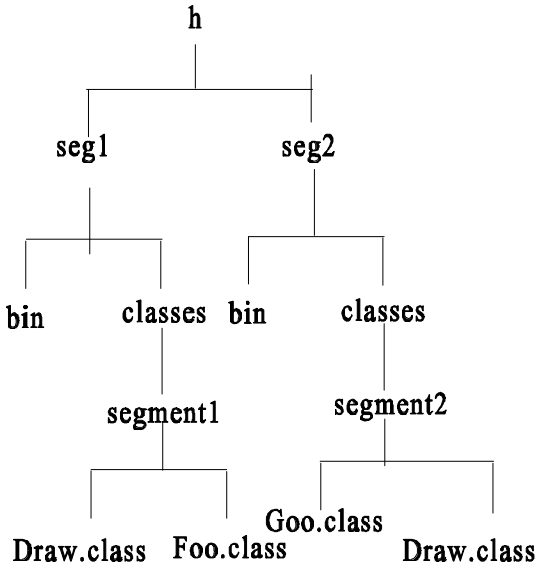
In UNIX the classpath would be set as:
setenv CLASSPATH = c:/h/seg1/classes; c:/h/seg2/classes

In Windows NT/95:
set CLASSPATH = c:\h\seg1\classes; c:\h\seg2\classes

```
//import all classes in the segment1 and segment2 class packages
import segment1.*;
import segment2.*;

//declaring and instantiating using full class specification to avoid name conflicts
segment1.Draw draw_obj1 = new segment1.Draw();
segment2.Draw draw_obj2 = new segment2.Draw();

// because there is not a naming conflict between Foo and Goo we don't have
// to use the full class specification. The classpath and import statements combine to enable the
```




```
//following unambigous abbreviated references. But fuller qualification would help preclude
//future problems.
```

```
Foo myfoo = new Foo();
Goo yourgoo = new Goo();
```

The difference between this code and the first example occurs within lines 5 - 7. With the segment name now used within the directory structure we can provide a unique name for each class. The class name maps to the subdirectories under the directories defined by CLASSPATH. The use of the segment name within the directory structure is **mandatory** for the COE.

To extend the above naming system we could also use the segment's unique prefix within the class library. The prefix is a unique descriptor for the segment provided by configuration management (CM).

Code example:

In UNIX the classpath would be set as:

```
setenv CLASSPATH = c:/h/seg1/classes; c:/h/seg2/classes
```

In Windows NT/95:

```
set CLASSPATH = c:\h\seg1\classes; c:\h\seg2\classes
```

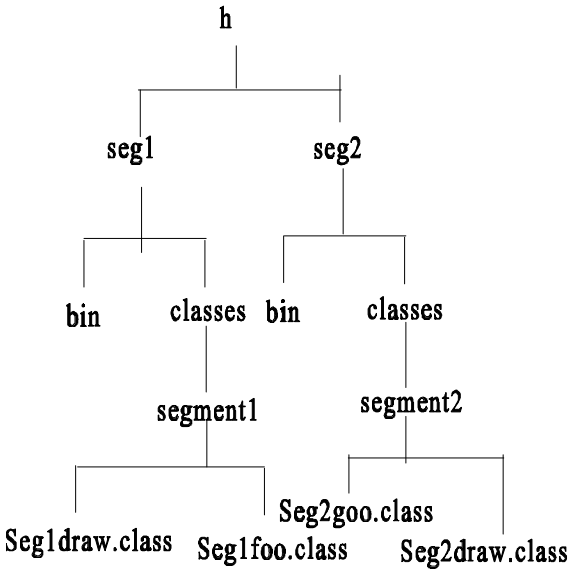
```
//import all classes in the Segment1 and Segment2 class packages
import segment1.*;
import segment2.*;

//declaring and instantiating
Seg1draw draw_obj1 = new Seg1draw();
Seg2draw draw_obj2 = new Seg2draw();
Seg1foo myfoo = new Seg1foo();
Seg2goo yourgoo = new Seg2goo();
```

The directory/files system would look like:

In lines 5 - 9 we once again declared and instantiated new objects. This time we only had to use the class name because they were unique. The compiler finds Seg1draw.class using both classpath (c:/segment1/classes) and then import (import segment1.*).

Why not simply use the segment prefix instead of the segment name in the directory structure to create a unique class name? As systems grow, segments may have several class libraries contained within them. The segment prefix while unique between segments would not be unique between the classes contained within a segment. For example the segment prefix UB is used to cover at several segments;including IFL, TMS, and UCP. The use of the segment prefix with the class name is **optional**. Developers are cautioned that naming conflicts within a segment are still possible and should be avoided.



1.Diskin, David. “Introduction To JAVA”. DISA/JEXFAdvanced Tech. Branch: http://spider.osfl.disa.mil/dii/atd/Java_brief/index.htm

2. Flanagan, David. JAVA in a Nutshell. Cambridge: O’Reilly, 1997, P. 17-18

3.Koosis, Donald, and David Koosis. JAVA Programming for Dummies. Foster City: IDG Books Worldwide Inc., 1997, P. 143

4.Lemay, Laura and Charles Perkins. teach yourself JAVA in 21 days. Indianapolis: Samsnet, 1996, P. 327